University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

A Color Sequencer

Review the Six-Step Process

Recall our six-step process for FSM Design:

1. develop an abstract model
2. specify I/O behavior
3. complete the specification
4. choose a state representation
5. calculate logic expressions
6. implement with flip-flops and gates

Let's Build a Color Sequencer

Let's do another example.

Let's build a color sequencer that cycles through a set of colors.

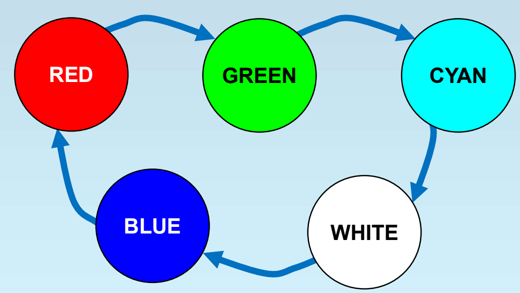
Imagine that we have an LED light that can output eight colors…

Our FSM will drive this light using the RGB signals.

RGB	color
000	black
001	blue
010	green
011	cyan
100	red
101	violet
110	yellow
111	white

Abstract Model for a Color Sequencer Has Five States

1. Our abstract model? A counter that goes through five colors. Like this:



Next, Define Inputs and Outputs

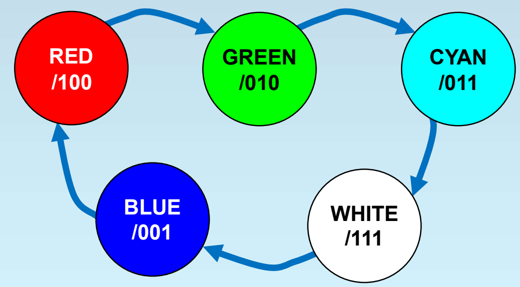
2. Inputs: none (it's a counter).

   Outputs? We can just read RGB from the table for each state.

RGB	color
000	black
001	blue
010	green
011	cyan
100	red
101	violet
110	yellow
111	white

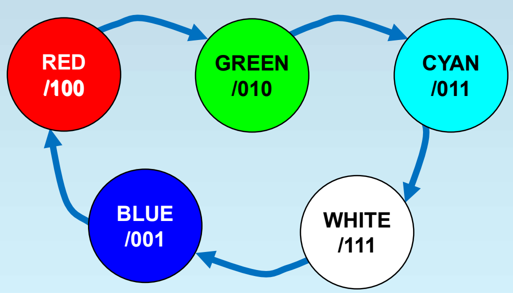
Outputs Represent Red, Green, and Blue

Let's add the outputs (as /RGB) to the states.



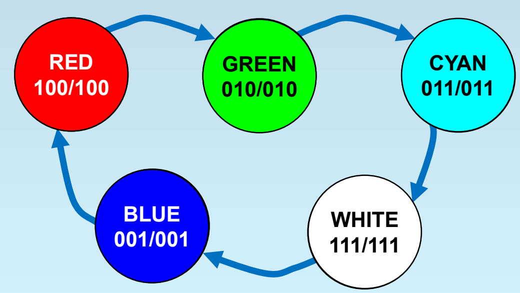
Completing the Specification

3. No inputs, so … specification is complete!



Use Unique Outputs as the Internal State IDs

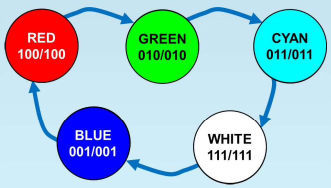
4. Outputs are again unique, so use them as state IDs as well.



Write a Next-State Table

S_2	S_1	S_0	S_2^+	S_1^+	S_0^+
0	0	0	x	x	x
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	x	x	x
1	1	0	x	x	x
1	1	1	0	0	1

5. Time for equations. Start by writing a next-state table.



Now Use K-Maps to Express the Next-State Values

S_2	S_1	S_0	S_2^+	S_1^+	S_0^+
0	0	0	x	x	x
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	x	x	x
1	1	0	x	x	x
1	1	1	0	0	1

Now copy into K-maps.

$S_0^+ = S_1$

S_0^+

S_2 \ S_1S_0	00	01	11	10
0	x	0	1	1
1	0	x	1	x

Now Use K-Maps to Express the Next-State Values

S_2	S_1	S_0	S_2^+	S_1^+	S_0^+
0	0	0	x	x	x
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	x	x	x
1	1	0	x	x	x
1	1	1	0	0	1

Now copy into K-maps.

$$S_1^+ = S_2'S_1 + S_2S_1'$$
$$= S_2 \oplus S_1$$

S_1^+

S_2 \ S_1S_0	00	01	11	10
0	x	0	1	1
1	1	x	0	x

Now Use K-Maps to Express the Next-State Values

S_2	S_1	S_0	S_2^+	S_1^+	S_0^+
0	0	0	x	x	x
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	x	x	x
1	1	0	x	x	x
1	1	1	0	0	1

Now copy into K-maps.

$$S_2^+ = S_2'S_0 = (S_2 + S_0')'$$

S_2^+

S_2 \ S_1S_0	00	01	11	10
0	x	1	1	0
1	0	x	0	x

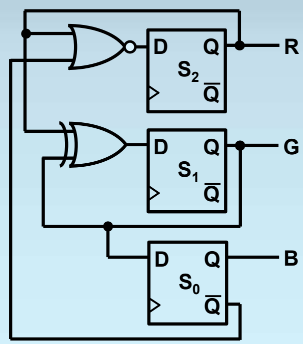
Implement Using Three Flip-Flops and Two Gates

6. Finally, we can implement, as shown to the right.

$S_2^+ = (S_2 + S_0')'$

$S_1^+ = S_2 \oplus S_1$

$S_0^+ = S_1$



Ready to Build It?

Are you excited?

Imagine that you go get your protoboard out.

You go to the lab.

You build the color sequencer.

You hook it to the LED light.

You turn it on.

…

It stays black.

Seem familiar?

Behavior Seems to Be Inconsistent

You debug for a while.

You play with wires.

You look at datasheets.

Everything seems right.

Sometimes it works.

Sometimes it flashes yellow or violet, then works.

Sometimes it stays black.

What's going on?

Our Don't Cares Become 0s for S_2

What happened to the "don't care" states?

Let's take a look.

We can use our K-maps or our equations.

For S_2, the x's became 0s.

S_2^+

S_2 \ S_1S_0	00	01	11	10
0	x	1	1	0
1	0	x	0	x

000 → 0??
101 → 0??
110 → 0??

One x Becomes a 1 for S_1

For S_1, the x for state 101 became a 1, and the others became 0s.

S_1^+

S_2 \ S_1S_0	00	01	11	10
0	x	0	1	1
1	1	x	0	x

000 → 00?
101 → 01?
110 → 00?

One x Becomes a 1 for S_0

For S_0, the x for state 110 became a 1, and the others became 0s.

So what comes after 000 (black)?

Black again!

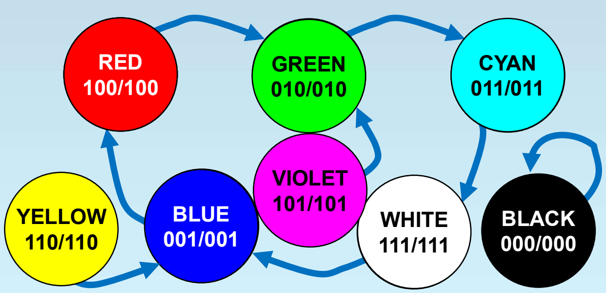
S_0^+

S_2 \ S_1S_0	00	01	11	10
0	x	0	1	1
1	0	x	1	x

000 → 000
101 → 010
110 → 001

Full Transition Diagram Illustrates Buggy Behavior

We can add these states to our diagram.



Avoid Bad States by Initializing the Counter State

What can we do? Let's add a way to initialize.

We can…

- **choose a specific (hardwired) initial state** at power-on (one from our loop*),
- **use muxes** to enable ourselves to set the state arbitrarily at any time,
- or use one signal to force the system into the loop, such as $\mathbf{S_0^+ = (S_1\text{'INIT'})'}$ (active low).

*Forcing all flip-flops to 0 doesn't help!

One Can Always Backtrack in the Design Process

Alternatively,

- we can go back to our K-maps and add loops.
- We may need to iterate a couple of times to find a design that always works.

We could also just choose specific next states for the states outside of our loop.

These approaches require more logic.